2 CLOSED-LOOP TRACKER

2.1 Introduction

We approach closed-loop trackers from the perspective of servomechanisms (servos) because that is what they are. More specifically, the closed loop trackers in radars are unity feedback, control loops or servos. We call them unity feedback because the predicted target parameter has the same units as the actual target parameter. Block diagrams for this type of servo for analog and digital trackers are shown in Figure 2. As shown, they have the same components but the filter and controlled elements are in different domains, as are the signals. In the analog servo, the signals are in the time domain and the filter and controlled element transfer functions are in the s-domain. In the digital tracker, the signals are in the discrete time domain and the filter and controlled element transfer functions are in the z-domain. In some cases, the tracker can be hybrid where, for example, the track filter might be digital and the controlled element might be analog. As a note, the type of receiver and signal processor (analog, digital, or hybrid) does not enter into determining whether the tracker is analog, digital or hybrid.

Figure 2 – Analog and Digital Closed Loop Tracker Block Diagrams
An example of an analog tracker would be the angle tracker in a single-target track radar with a dish antenna. In this case, the controlled element would be the controllers and motors that move the dish, thus the controlled element would be analog. We could also treat the dynamics of the dish controller as part of the track filter since they influence the closed loop response. Unless the radar is really old, the rest of the track filter would consist of some op-amp-based R-C (resistor-capacitor) networks and this would also be analog. Thus, the overall tracker would be considered an analog tracker and we would analyze it using differential equations and s-domain transfer functions.

An example of a digital tracker would be the angle tracker in a radar that uses a phased array antenna and tracks multiple targets, or performs multiple functions such as track, search, missile guidance, etc. Because of this multiplexing, the radar produces error signals, for any one target, at discrete instances of time. Also, in such radars the track filter would most likely be implemented in a digital computer, possibly as a g-h, g-h-k or Kalman filter. The controlled element would be the beam steering computer and the phase shifters in the antenna, which move the beam at discrete instances of time. Since events in this tracker are occurring at discrete instants of time, we would consider it a digital tracker and analyze it using difference equations and z-transforms.

An example of a hybrid tracker might be the angle tracker in a seeker that implements the track filter in a digital computer as a z-transfer function or a g-h, etc. track filter. However, the antenna controller might be analog if the seeker has a dish antenna. Also, we consider that the seeker is tracking a single target and is not performing multiple functions. The fact that the track filter is implemented as a digital filter means that we should analyze it using difference equations and z-transforms. However, the fact that the controller for the antenna is analog means that we should analyze it using differential equations and s-transforms. Actually, what we want to do is recast the z-transfer functions to the s-domain or recast the s-transfer functions to the z-domain and analyze the overall system in a single domain. That decision is usually determined by designer/analyst preferences. However, it could be determined by the overall operation of the hybrid tracker. For example, if the tracker updates the antenna controller at fixed, discrete instances of time, the analyst might be inclined to treat it as a digital tracker since events are happening at discrete instants of time.

In the block diagrams of Figure 2 we represented the radar as a summer, actually a difference, and gain, $K_R$. From the track loop perspective, that is all the radar does. That is, it forms the error between the target parameter and its estimate, and outputs a voltage or number that is proportional to the error. In practice, the gain, $K_R$, is really a non-linear curve that is termed a discriminator curve. When we study how the radar forms the error signal, we will see how this non-linear curve is derived.

In many cases the controlled element can be considered a units converter. Its input is a voltage or number and its output is the parameter being tracked. This holds even if the controlled element is a hardware device such as a motor that moves the antennas, or a timing circuit that positions a range gate. In both cases, the input is a voltage or number and the output is angle for the antenna or time for the timing circuit.

We will begin our study by considering analog trackers in more detail.
2.2 Analog Trackers

Most analog trackers are Type 1 or Type 2 servomechanisms (servos). As you may recall from control theory, a Type 1 servo tracks a constant (step) input with zero steady state error and a Type 2 servo tracks linearly varying (ramp) and constant inputs with zero steady state error. These features are attractive for radar trackers in that we would like to have zero steady state tracking error.

Figure 3 contains a block diagram of an analog tracker. As we discussed earlier, for now, we represent the radar by a summer and a gain $K_R$. Also, we have combined the controlled element dynamics with the track filter and renamed the combination $G_{OL}(s)$, where the subscript $OL$ stands for open loop. Further, we replaced the controlled element by $1/K_R$ to emphasize that its purpose, in our block diagram, is to convert from voltage back to the units of the parameter being tracked.

The system type (Type 1, Type 2, etc.) is determined by the number of poles of $G_{OL}(s)$ that are located at the origin of the $s$-plane (at $s=0$). If $G_{OL}(s)$ has one pole at the origin, the system is Type 1, if it has two poles at the origin, it is Type 2, if it has $N$ poles at the origin, it is Type $N$. Sometimes poles at the origin occur naturally because of the properties of the controlled element. In other cases, the filter is designed to include poles at the origin.

An example of where the controlled element contributes a pole at the origin is where a DC motor is used to control the position of a reflector antenna. To see how this occurs, we consider the simplified schematic of a DC motor shown in Figure 4. For such a motor, the rotational speed, $\Omega(s)$, is related to the armature current by

$$\Omega(s) = \frac{K}{s+a} I(s)$$

where the $K/(s+a)$ term accounts for the rotor inertia, load inertia and other such factors. Further, from circuit theory,
\[ V(s) = (R + sL)I(s). \]  \hspace{1cm} (2)

Finally, we know that rotation rate, \( \omega(t) \), is related to angle, \( \theta(t) \), by

\[ \omega(t) = \dot{\theta}(t) \]  \hspace{1cm} (3)

and thus that

\[ \Omega(s) = s\Theta(s) - \theta(0) \]  \hspace{1cm} (4)

Or

\[ \Omega(s) = s\Theta(s) \]  \hspace{1cm} (5)

if we assume an initial angle of \( \theta(0) = 0 \). In the above, \( \theta(t) \) and \( \Theta(s) \) represent the angular position of the motor shaft.

We can solve for \( \Theta(s) \) in terms of \( V(s) \) as

\[ \Theta(s) = \frac{\Omega(s)}{s} = \frac{K}{s(s + a)}I(s) = \frac{K}{s(s + a)(R + sL)}V(s). \]  \hspace{1cm} (6)

and use this to write the transfer function of the motor as

\[ G_m(s) = \frac{\Theta(s)}{V(s)} = \frac{K}{s(s + a)(R + sL)}. \]  \hspace{1cm} (7)

From this we note that the transfer function of the motor has a pole at the origin of the s-plane. Thus, \( G_{OL}(s) \) will have at least one pole at the origin and the tracker will be at least a Type 1 servo. If the track filter was designed so that it also had one pole at \( s = 0 \), \( G_{OL}(s) \) would have two poles at the origin of the s-plane and the tracker would be a Type 2 servo.

Analog angle trackers are typically Type 1 servos in ground-based radars. It appears as if the logic behind this is that the target angle changes slowly and that such changes will cause tolerable angle track errors for the case of a constantly varying angle. Recall that a Type 1 servo has zero error for a constant input and a constant, or bias, error for constantly changing input. It is likely Type 2 trackers would be used in air-borne radars and missile seekers since motion of the aircraft or seeker could result in fairly large angle rates.
Range trackers tend toward Type 2 servos. The logic here is that we expect range to be changing at a rate that may not be small. Because of this, a Type 2 servo is more attractive because it will follow a ramp (constantly changing) input with zero steady-state error. If the range changes with a constant acceleration profile, the track error will be a constant. If the acceleration is not large, the magnitude of the error may be tolerable.

Doppler trackers are typically Type 0 or Type 1 servos. If the Doppler tracker uses a simple frequency locked loop, it could be a Type 0 servo. If the Doppler tracker uses a phase locked loop, it will most likely be a Type 1, or even a Type 2, servo.

Analog trackers do not normally use higher than Type 2 servos because it is difficult to build stable analog trackers that are Type 3 or higher. However, as we will see, digital trackers that use g-h filters are Type 2 servos and those that use g-h-k filters are Type 3 servos.

The above discussions imply that angle, range or Doppler change with a constant rate or acceleration. In practice, neither is true. This means that the above conclusions about how the tracker will behave do not strictly hold. Also, the trackers will be influenced by noise in the radar (the error sensor), which will further cloud the interaction between servo type and errors. In spite of this we still discuss servo types and their relation to behavior in terms of steady state error.

2.2.1 System Type and Steady State Error

We will now take some time to review how system type determines steady state tracking error. Since we are interested in tracking error, we will be concerned with the term $\Delta x(t)$ in Figure 3. We will work in the s-domain.

From Figure 3 we have

$$\Delta X(s) = X(s) - \hat{X}(s)$$  \hspace{1cm} (8) \hspace{1cm} or

$$\hat{X}(s) = X(s) - \Delta X(s).$$  \hspace{1cm} (9) \hspace{1cm}

We further note that

$$\hat{X}(s) = (1/K_r) G_{OL}(s) K_r \Delta X(s) = G_{OL}(s) \Delta X(s).$$  \hspace{1cm} (10) \hspace{1cm}

Combining (10) and (9) gives

$$X(s) - \Delta X(s) = G_{OL}(s) \Delta X(s)$$  \hspace{1cm} (11) \hspace{1cm} and solving for $\Delta X(s)$ in terms of $X(s)$ gives

$$\Delta X(s) = \frac{1}{1 + G_{OL}(s)} X(s).$$  \hspace{1cm} (12) \hspace{1cm}

For the next step we want to specifically denote the poles of $G_{OL}(s)$ that are at the origin of the s-plane. We do this by writing $G_{OL}(s)$ as
\[ G_{ol}(s) = \frac{G(s)}{s^N} \]  

where \( G(s) \) has no poles (or zeros) at \( s = 0 \). Substituting (13) into (12) yields

\[ \Delta X(s) = \frac{1}{1 + G(s)} s^N X(s) = \frac{s^N}{s^N + G(s)} X(s). \]  

We want to examine the steady state value of the error, \( \Delta X(s) \), for different values of \( N \) and inputs of the form

\[ x(t) = A \frac{t^M}{M!} U(t) \]  

where \( M \) is an integer and \( U(t) \) is the unit step function and is defined as

\[ U(t) = \begin{cases} 0 & t \leq 0 \\ 1 & t > 0 \end{cases}. \]  

\( x(t) \) is a step input for \( M = 0 \), a ramp input for \( M = 1 \) and a quadratic, or constant acceleration, input for \( M = 2 \).

Taking the Laplace transform of (15) gives

\[ X(s) = \frac{A}{s^{M+1}}. \]  

We find the error for the class of inputs defined by (15) and (17) by substituting (17) into (14). The result is

\[ \Delta X(s) = \frac{s^N}{s^N + G(s)} \frac{A}{s^{M+1}}. \]  

To find the steady state value of \( \Delta x(t) \) we invoke the final value theorem of Laplace transforms and write

\[ \Delta x_{SS} = \lim_{s \to 0} \left[ s \Delta X(s) \right] = \lim_{s \to 0} \left[ \frac{s^N}{s^N + G(s)} \frac{A}{s^{M+1}} \right]. \]  

We will consider a few of examples and then generate a table for the cases of Type 0, 1 and 2 servos with step, ramp and quadratic inputs. For the first example we consider a Type 0 servo with a step input. For this case we have \( N = 0 \) and \( M = 0 \). With this, (19) reduces to

\[ \Delta x_{SS} = \lim_{s \to 0} \left[ \frac{s^{0+1}}{s^0 + G(s)} \frac{A}{s^{0+1}} \right] = \lim_{s \to 0} \left[ \frac{A}{1 + G(s)} \right] = \frac{A}{1 + G(0)} \]  

which tells us that a Type 0 servo with a constant input (step input) will have a steady-state error. Further, the steady state error will be directly proportional to the magnitude of the step input and
inversely proportional to the DC gain of the open loop transfer function, with the poles at the origin removed. The consequence of the last statement is that the steady state error can be controlled by increasing the DC gain of the tracker. Unfortunately, this can have a deleterious effect on the transient performance of the tracker.

If we consider a ramp input to a Type 0 servo we would have \( N = 0 \) and \( M = 1 \). This would give a steady state error of

\[
\Delta x_{ss} = \lim_{s \to 0} \left[ \frac{s^{0+1}}{s^0 + G(s)} A \right] = \lim_{s \to 0} \left[ \frac{A}{s(1+G(s))} \right] = \infty. \tag{21}
\]

In other words, the steady state error would be infinite for this case.

As still another example, we consider a step input to a Type 1 servo. For this case we would have \( N = 1 \) and \( M = 0 \) and

\[
\Delta x_{ss} = \lim_{s \to 0} \left[ \frac{s^{1+1}}{s^1 + G(s)} A \right] = \lim_{s \to 0} \left[ \frac{sA}{s + G(s)} \right] = 0 \tag{22}
\]

or zero steady state error. That is, a Type 1 servo will, in steady state, perfectly track a constant input.

We can generalize the above discussions to generate Table 1 which is a tabulation of steady state errors for Type 0, 1 and 2 servos with step, ramp and quadratic inputs.

<table>
<thead>
<tr>
<th>Servo Type</th>
<th>Input Type</th>
<th>Step</th>
<th>Ramp</th>
<th>Quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \frac{A}{1+G(0)} )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( \frac{A}{G(0)} )</td>
<td>( \infty )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>( \frac{A}{G(0)} )</td>
<td></td>
</tr>
</tbody>
</table>

The above discussions provided a characterization of the steady state error. It turns out that if the tracker is too sluggish, it could lose track before it reaches steady state. Thus, the fact that it could have, say, zero steady state error becomes a moot point. We will see this in some later examples. For now we want to discuss how \( G_{ol}(s) \) affects the transient behavior of a closed loop tracker.

2.2.2 Root Locus
The easiest way to get a qualitative idea of the transient behavior of an analog tracker is via the root locus, which gives an idea of how the closed loop poles of the tracker vary as the open loop gain varies from zero to infinity. In general, it provides an idea of whether the tracker will go unstable and a general idea of whether the closed loop transfer function will have all real poles or some complex poles. Complex poles are a requirement if we want to have an underdamped transient response, which is generally the case for radar trackers.

We will restrict ourselves to unity feedback systems since this is the type of servo that we have indicated radar trackers use. To study root locus, we will simplify the tracker block diagram to the form shown in Figure 5. We have eliminated the $K_R$ and $1/K_R$ blocks and expanded the open loop transfer function to denote the open loop gain, $K_{OL}$ and the poles and zeros of the open loop transfer function.

![Figure 5 – Block Diagram Used for Root Locus Discussions](image)

To set the stage for a root locus algorithm, we want to find the closed loop transfer function of the tracker. Specifically, we want to find

$$G_{cl}(s) = \frac{\hat{X}(s)}{X(s)}.$$  \hspace{1cm} (23)

If we combine (8) and (12) and perform the appropriate manipulations we get

$$\hat{X}(s) = X(s) - \frac{1}{1 + G_{ol}(s)} X(s) = \frac{G_{ol}(s)}{1 + G_{ol}(s)} X(s)$$  \hspace{1cm} (24)

and

$$G_{cl}(s) = \frac{G_{ol}(s)}{1 + G_{ol}(s)},$$  \hspace{1cm} (25)

or using the $G_{ol}(s)$ equation from Figure 5,

$$G_{cl}(s) = \frac{K_{ol} (s - z_1)(s - z_2)\cdots(s - z_m)}{(s - p_1)(s - p_2)\cdots(s - p_n) + K_{ol} (s - z_1)(s - z_2)\cdots(s - z_m)}.$$  \hspace{1cm} (26)

We note that as $K_{ol}$ approaches zero, the poles of $G_{cl}(s)$ will equal the poles of $G_{ol}(s)$ and as $K_{ol}$ approaches infinity, the poles of $G_{cl}(s)$ will approach the zeros of $G_{ol}(s)$. This means
that poles of $G_{cl}(s)$ will start at the poles of $G_{ol}(s)$ and terminate on the zeros of $G_{ol}(s)$ as $K_{ol}$ varies from zero to infinity. If $n > m$, i.e. $G_{ol}(s)$ has more poles than zeros, the usual case, we assume that there are $n - m$ zeros of $G_{ol}(s)$ at infinity. This means that some of the locus branches will go to infinity. The zeros of $G_{cl}(s)$ are the zeros of $G_{ol}(s)$. With this as a starting point, we can present an algorithm for sketching a root locus as\textsuperscript{1}:

1. Draw an s-plane where the horizontal axis is the real part of $s$ and the vertical axis is the imaginary part of $s$.
2. Plot the poles of $G_{ol}(s)$ as $\times$’s and the zeros of $G_{ol}(s)$ as o’s.
3. Plot the real-axis portion of the root locus using the rule that a point will be on the locus if there are an odd number of poles and zeros to the right of the point.
4. The locus always migrates away from the poles of $G_{ol}(s)$ and towards the zeros of $G_{ol}(s)$.
5. If $G_{ol}(s)$ has more poles than zeros, the locus will go to infinity.
6. If two real-axis branches of the locus meet, the locus will branch away from the real axis. Sometimes the locus heads toward infinity and other times it returns to the real axis.
7. If the locus branches away and goes to infinity, it will approach infinity along asymptotes that are located at angles of

$$\theta_k = (2k - 1)\frac{180}{K} \deg \quad k = 1, 2 \ldots K$$

where $K = n - m$, the difference of the number of poles and zeros of $G_{ol}(s)$.

8. The asymptotes intersect the real axis at the location

$$s_{asy} = \frac{\sum_{i=1}^{n} p_i - \sum_{j=1}^{m} z_j}{K}.$$  \hfill (28)

Let us consider a few examples to illustrate how to construct a root locus. For the first case we consider

$$G_{ol}(s) = \frac{K_{ol}}{s + a}$$ \hfill (29)

where $a \geq 0$. For this case there will be one pole of $G_{ol}(s)$ at $s = -a$ and no zeros. Since $K = n - m = 1$ there will be one asymptote at

$$\theta_1 = (2 - 1)\frac{180}{1} \deg = 180' .$$ \hfill (30)

Since there are an even number (zero) of poles and zeros to the right of $s > -a$, there will be no locus on the real axis for $s > -a$. However, since there are an odd number of poles and zeros

\textsuperscript{1} The derivation of this algorithm can be found in most text books on classical control theory and will not presented here.
(one) to the right of $s \leq -a$ there will be a locus in this region. Finally, the locus starts on the pole at $s = -a$ and goes to infinity along the asymptote at $\theta_1 = 180^\circ$. The resultant root locus is shown in Figure 6.

![Figure 6 – Root locus for Equation (29)](image)

The root locus of Figure 6 tells us the closed loop response for a step input will be of the form

$$x(t) = c + de^{-\alpha t}$$

(31)

where $\alpha$ depends on the location of the closed loop pole. Since $\alpha$ increases with $K_{OL}$ the settling time of $x(t)$ will decrease. Said another way, as $K_{OL}$ increases the settling time of the tracker will decrease. Since the closed loop pole, $-\alpha$, always remains in the left half of the $s$-plane, the tracker will always be stable.

For a second example we consider the open loop transfer function

$$G_{OL}(s) = \frac{K_{OL}}{(s + a)(s + b)}$$

(32)

where $b > a \geq 0$. A sketch of the root locus for this case is shown in Figure 7. Since there are an odd number of poles and zeros to the right of the region between the two open loop poles, the locus will exist in this region. For all other points on the real axis of the $s$-plane, there are an even number of poles and zeros to the right, thus there is no locus at these points. Since the locus must move away from the poles, segments of the locus start at $s = -a$ and $s = -b$ and meet somewhere between the two poles. The locus then branches away from the real axis and approaches asymptotes located at
\[ \theta_k = (2k - 1) \frac{180}{2} \text{deg} \quad k = 1, 2 \text{ or } 90^\circ \text{ and } 270^\circ. \]  \hspace{1cm} (33)

There are methods of computing the exact point where the locus breaks away from the real axis, but the equations are somewhat tedious and not worth computing for qualitative analyses.

Figure 7 – Root Locus for Equation (32)

The root locus of Figure 7 tells us that, for low values of \( K_{OL} \) the time, or transient, response will be overdamped because \( G_{cl}(s) \) will have two real poles. One will be located between the pole at \( s = -a \) and the breakaway point and the other will be located between \( s = -b \) and the breakaway point. At some value of \( K_{OL} \), the poles will become equal and the transient response will be critically damped. Finally, at larger values of \( K_{OL} \) the poles will become complex and the transient response will become underdamped. Examples of these three types of time response are contained in Figure 8. As \( K_{OL} \) continues to increase, the frequency of the underdamped oscillation will increase because the imaginary parts of the poles will become larger. However, the damping time will remain fixed because the real parts of the poles stay constant at some value between \(-a\) and \(-b\) (the breakaway point).
For a third example we consider a Type 1 servo with an open loop transfer function of

$$G_{OL}(s) = \frac{K_{OL}(s+b)}{s(s+a)}.$$  \hfill (34)

We consider two cases for the relative sizes of $a$ and $b$. For the first we let $a > b > 0$. The resulting root locus is shown in Figure 9. As can be seen there will be locus points on the real axis between the pole at the origin and the zero at $s = -b$ and for $s \leq -a$. Since $K = n - m = 2 - 1 = 1$ there will be one asymptote at 180 degrees. As $K_{OL}$ increases one of the closed loop poles moves toward the open loop zero and the other moves toward infinity. Since the two closed loop poles are real, we would expect the closed loop time response to be overdamped. However, the presence of the closed loop zero at $s = -b$ (recall that the closed loop zeros are the open loop zeros) will affect the time response and could make it appear to be underdamped.
For the second case we let $b > a > 0$. The resulting root locus is shown in Figure 10. In this case, the locus will exist on the real axis between the poles at the origin and $s = -a$, and to the left of the zero at $s = -b$. However, in this case, the locus leaves the poles at the origin and $s = -a$ and come together somewhere between the two poles. The locus then branches away from the real axis and returns to the real axis somewhere to the left of the zero at $s = -b$. Finally, the locus travels to the zero and to infinity along the asymptote at 180 degrees.

This case is interesting in that it says the two closed loop poles are real and unequal for small values of $K_{OL}$, and thus that the transient response is overdamped (except for the possible influence of the closed loop zero at $s = -b$). For intermediate values of $K_{OL}$ the poles become complex, which yields an underdamped response. As the response becomes underdamped the frequency of oscillation will increase and then decrease while the settling time will continue to
decrease. As the gain is further increased, the poles again become real and the response becomes overdamped again.

The fact that the locus follows a circle between leaving and re-entering the real axis is notional. To determine the actual shape we would need to compute and plot the poles of $G_{cl}(s)$ as we vary $K_{ol}$. However, for qualitative analyses this is not necessary.

As a final example we consider the open loop transfer function

$$G_{ol}(s) = \frac{K_{ol}}{s(s+a)(s+b)} \tag{35}$$

with $b > a > 0$. The resultant root locus is shown in Figure 11. As with the previous example, the real axis locus exists between the pole at the origin and the pole at $s = -a$ and to the left of the pole at $s = -b$. Since $K = n - m = 3$ the locus will approach infinity along three asymptotes located at 60, 180 and 300 degrees. Further, the locus approaches the asymptotes at 60 and 300 degrees after it branches away from the real axis somewhere between the origin and $s = -a$. We can conclude that the response will start as an overdamped response for small values of open loop gain and then become underdamped for larger values of open loop gain. As the open loop gain continues to increase, two of the closed loop poles move into the right half of the s-plane, which means that the tracker will become unstable.

![Figure 11 – Root Locus of Equation (35)](image)

2.2.3 Transient Response

While root locus is handy for performing qualitative analyses of the expected transient performance of a tracker, it is not very useful in designing track filters to achieve a specific desired closed loop response. For that case we take a more direct approach of relating the poles and zeros of the track filter (actually the combination of the track filter and controlled element – recall that we combined them). We will not attempt to create a general approach but will look at specific cases. In particular, we look at two cases that yield second-order closed loop responses.
The first is the example of (32), which can be adjusted to yield a Type 0 or Type 1 servo and a response that can vary between over damped and underdamped. The second is the example of (34), which can be adjusted to yield a Type 0, 1 or 2 servo with a response that can also vary between over damped and underdamped.

For the first example, we adjust (32) so that it yields a Type 1 servo by setting \( b = 0 \). If we use (32) in (25), with \( b = 0 \), we have

\[
G_{OL}(s) = \frac{K_{OL}}{(s + a)(s + b)} \bigg|_{b=0} = \frac{K_{OL}}{s(s + a)} \tag{36}
\]

and

\[
G_{CL}(s) = \frac{G_{OL}(s)}{1 + G_{OL}(s)} = \frac{\frac{K_{OL}}{s(s + a)}}{1 + \frac{K_{OL}}{s(s + a)}} = \frac{\frac{K_{OL}}{s(s + a)}}{s(s + a) + K_{OL}} = \frac{K_{OL}}{s^2 + as + K_{OL}}. \tag{37}
\]

We can write this in a standard quadratic form as

\[
G_{CL}(s) = \frac{K_{OL}}{s^2 + as + K_{OL}} = \frac{K_{OL}}{s^2 + 2\zeta\omega_n s + \omega_n^2}. \tag{38}
\]

In the far right side of this equation, \( \omega_n \) is termed the undamped natural frequency and \( \zeta \) is termed the damping coefficient or damping ratio. These parameters tell us something about the settling time and the oscillation frequency of the transient.

If we factor the denominator of the last term of (38) we get roots located at

\[
s_1, s_2 = -\zeta\omega_n \pm \omega_n \sqrt{\zeta^2 - 1}. \tag{39}
\]

If \( \zeta > 1 \) both roots are real, which means that the transient response will be overdamped. If \( \zeta = 1 \) the two roots are equal and we say we have a critically damped response. If \( \zeta < 1 \) the roots are complex conjugates and the response is underdamped. (By the way, we can see all three of these conditions by examining the root locus for this example.) For the underdamped case, the closed loop impulse response of the tracker will be of the form

\[
g_{CL}(t) = \mathcal{L}^{-1}[G_{CL}(s)] = \mathcal{L}^{-1}\left[\frac{\frac{K_{OL}}{s^2 + 2\zeta\omega_n s + \omega_n^2}}{s^2 + \omega_n^2}\right] = Ae^{-\omega_d t} \sin \omega_d t U(t) \tag{40}
\]

where \( \mathcal{L}^{-1}[F(s)] \) denotes the inverse Laplace transform of \( F(s) \). In (40), \( \omega_d = \omega_n \sqrt{1 - \zeta^2} \) is the damped frequency of the response. It, and not \( \omega_n \), is the frequency of the ringing in the transient response. The time constant, or decay time constant, is \( 1/\zeta\omega_n \). It determines how long the response will take to decay. Generally, we select \( \zeta \) to be a value between 0.5 and \( 1/\sqrt{2} \approx 0.707 \) and then choose \( \omega_n \) to give a desired decay time. As we will see in a homework problem, \( \zeta = 0.707 \) results in a step response that has one overshoot peak before reaching steady state. When \( \zeta = 0.5 \), the rise time of the response will be faster but there will be more ringing as the
response settles (assuming \( \omega_n \) has been chosen to keep the same decay time). As \( \zeta \) gets smaller, the rise time decreases and the ringing increases. As \( \zeta \) becomes larger, the rise time increases and the ringing disappears (the response becomes critically damped and then overdamped).

In some cases, actually, most of the time, we find it intuitively more appealing to characterize the closed loop behavior in terms of frequency rather than time. Specifically, we specify the damping ratio, \( \zeta \), and a closed loop bandwidth, \( \omega_k \) rad/s or \( f_c = \omega_k/2\pi \) Hz. Since \( G_{OL}(s) \) is a low pass response we define the bandwidth as the value of \( \omega_k \) at which \( |G_{OL}(s)|^2 \) is \( \frac{1}{2} \) its value at \( s = 0 \). In equation form, we want the \( \omega_k \) such that

\[
\left| G_{OL}(s) \right|^2 \bigg|_{s=j\omega_k} = \frac{1}{2} \left| G_{OL}(s) \right|^2
\]

or

\[
\left| \frac{K_{OL}}{-\omega_k^2 + 2j\omega_n\omega_c + \omega_n^2} \right|^2 = \frac{1}{2} \left| \frac{K_{OL}}{\omega_n^2} \right|^2
\]

or

\[
\frac{K_{OL}^2}{\left(\omega_n^2 - \omega_c^2\right)^2 + (2\zeta\omega_n\omega_c)^2} = \frac{K_{OL}^2}{2\omega_n^4}
\]

or

\[
\left(\omega_n^2 - \omega_c^2\right)^2 + (2\zeta\omega_n\omega_c)^2 = 2\omega_n^4.
\]

Solving (44) for \( \omega_n \) results in

\[
\omega_n = \omega_c \sqrt{\left(1 - 2\zeta^2\right) + 2\zeta^2 - 1}.
\]

Interestingly, if we choose \( \zeta = 1/\sqrt{2} \) we get \( \omega_n = \omega_c \) or that the undamped natural frequency is the closed loop bandwidth.

Once we determine \( \zeta \) and \( \omega_n \), we can use (38) to relate them to \( a \) and \( K_{OL} \) by equating like powers of \( s \). This gives

\[
a = 2\zeta\omega_n
\]

and

\[
K_{OL} = \omega_n^2.
\]

With this we get an open loop transfer function of

\[
G_{OL}(s) = \frac{\omega_n^2}{s(s + 2\zeta\omega_n)}
\]
For our second example we consider (34) with $a = 0$. This results in a Type 2 servo with a root locus that will look similar to Figure 10 except that the locus will break away from the real axis at $s = 0$. It will still follow the generally circular path indicated in Figure 10.

For this case we have

$$G_{cl}(s) = \frac{G_{ol}(s)}{1+G_{ol}(s)} = \frac{K_{ol}(s+b)}{s^2} = \frac{K_{ol}(s+b)}{s^2 + K_{ol}s + bK_{ol}}$$

(49)

which is also a second order response, with an added closed loop zero at $s = -b$. If we equate the denominator to the standard quadratic form we have

$$K_{ol} = 2\zeta\omega_n$$

(50)

and

$$b = \frac{\omega_n}{2\zeta}.$$  

(51)

This gives an open loop transfer function of

$$G_{ol}(s) = \frac{K_{ol}(s+b)}{s(s+a)} = \frac{2\zeta\omega_n}{s^2}$$

(52)

2.3 Modeling Analog Closed Loop Trackers

Now that we have a technique for determining the properties of analog trackers we want to consider how to simulate them. One method would be to use something like Matlab’s Simulink®. However, its use becomes cumbersome when we want to include the error sensor and target model to the simulation. Because of this, we will discuss two other approaches. In one we derive a state variable representation of $G_{ol}(s)$ and use a differential equation solver like Runge-Kutta, predictor-corrector or Matlab’s ode45 to model the system. The second method is more direct but requires a bilinear transform routine and the Matlab filter function.

2.3.1 State variable method

To use the state variable method we need to review how to find a state variable representation of an $s$-domain transfer function. We consider the transfer function

$$G(s) = \frac{Y(s)}{U(s)} = \frac{a_0 s^n + a_1 s^{n-1} + \cdots + a_n}{s^n + a_1 s^{n-1} + \cdots + a_n} = \text{transfer function}$$

(53)

which is $G_{ol}(s)$ without the $K_{ol}$ term. In this equation we stipulate that $m < n$ which will apply to the cases of interest to us. If $m = n$ we would need to add an extra step of dividing the numerator by the denominator before applying the technique below. If $m > n$ we have a
situation where the transfer function has derivatives, which we avoid because they are physically unrealizable.

There are several state variable representations that will result in the transfer function of (53). The particular form we will use is the phase variable form because it is the easiest to derive.

For the first step we divide (53) into the two equations

\[
W(s) = \frac{1}{s^n + a_1s^{n-1} + a_2s^{n-2} + \cdots + a_{n-1}s + a_n}
\]  

(54)

and

\[
Y(s) = s^m + b_1s^{m-1} + b_2s^{m-2} + \cdots + b_{m-1}s + b_m.
\]  

(55)

We solve (54) for \(U(s)\) in terms of \(W(s)\) and (55) for \(Y(s)\) in terms of \(W(s)\) to get

\[
(s^n + a_1s^{n-1} + a_2s^{n-2} + \cdots + a_{n-1}s + a_n)W(s) = U(s)
\]  

(56)

and

\[
Y(s) = (s^m + b_1s^{m-1} + b_2s^{m-2} + \cdots + b_{m-1}s + b_m)W(s).
\]  

(57)

We next use the Laplace transform of derivatives to convert (56) and (57) to differential equations. The particular Laplace transform of interest is

\[
\mathcal{L}\left[\frac{d^n f(t)}{dt^n}\right] = s^n F(s) - \sum_{i=0}^{n-1} s^{n-i-1} \frac{d^i f(t)}{dt^i}
\]  

(58)

where the summation captures the initial conditions on all of the derivatives of \(f(t)\). Since, the transfer function of a system is defined as the ratio of the Laplace transform of the output divided by the Laplace transform of the input, assuming the initial conditions are zero, we can ignore the initial conditions terms of (58) when finding the differential equations. With this, we can transform (56) and (57) to differential equations as

\[
\dot{w}^{(n)}(t) + a_1\dot{w}^{(n-1)}(t) + a_2\dot{w}^{(n-2)}(t) + \cdots + a_{n-1}\dot{w}(t) + a_nw(t) = u(t)
\]  

(59)

and

\[
\dot{w}^{(m)}(t) + b_1\dot{w}^{(m-1)}(t) + b_2\dot{w}^{(m-2)}(t) + \cdots + b_{m-1}\dot{w}(t) + b_mw(t) = y(t).
\]  

(60)

In these equations \(\dot{w}^{(k)}(t) = d^k w(t)/dt^k\).

To transform this to a state variable representation we define the state variables as
\[
\begin{bmatrix}
    x_1(t) \\
x_2(t) \\
    \vdots \\
x_n(t)
\end{bmatrix} = \begin{bmatrix}
w(t) \\
dw(t)/dt \\
    \vdots \\
    d^{n-1}w(t)/dt^{n-1}
\end{bmatrix}.
\]  

(61)

We next take the derivative of both sides of (61) to give
\[
\begin{bmatrix}
    \dot{x}_1(t) \\
    \dot{x}_2(t) \\
    \vdots \\
    \dot{x}_n(t)
\end{bmatrix} = \begin{bmatrix}
dw(t)/dt \\
d^2w(t)/dt^2 \\
    \vdots \\
d^n w(t)/dt^n
\end{bmatrix} \begin{bmatrix}
x_2(t) \\
x_3(t) \\
    \vdots \\
x_n(t)
\end{bmatrix}.
\]  

(62)

To get the right vector of (62) we made use of (61).

We next need to consider the last element of the right vector. From (50) we have
\[
w^{(n)}(t) = -a_1w^{(n-1)}(t) - a_2w^{(n-2)}(t) - \cdots - a_{n-1}w^{(1)}(t) - a_n w(t) + u(t)
\]  

(63)

or, with (61)
\[
w^{(n)}(t) = -a_1x_n(t) - a_2x_{n-1}(t) - \cdots - a_{n-1}x_2(t) - a_n x(t) + u(t)
\]  

(64)

Substituting this into (62) gives
\[
\begin{bmatrix}
    \dot{x}_1(t) \\
    \dot{x}_2(t) \\
    \vdots \\
    \dot{x}_n(t)
\end{bmatrix} = \begin{bmatrix}
x_2(t) \\
x_3(t) \\
    \vdots \\
\end{bmatrix} \begin{bmatrix}
    -a_1x_n(t) - a_2x_{n-1}(t) - \cdots - a_{n-1}x_2(t) - a_n x(t) + u(t)
\end{bmatrix}.
\]  

(65)

We can write this in matrix form as
\[
\dot{X} = AX + Bu(t)
\]  

(66)

where
\[
X = \begin{bmatrix}
x_1(t) \\
x_2(t) \\
x_3(t) \\
    \vdots \\
x_n(t)
\end{bmatrix},
\]  

(67)
\[
\dot{X} = \begin{bmatrix}
\dot{x}_1(t) \\
\dot{x}_2(t) \\
\dot{x}_3(t) \\
\vdots \\
\dot{x}_n(t)
\end{bmatrix},
\]

\[
A = \begin{bmatrix}
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1 \\
-a_n & -a_{n-1} & -a_{n-2} & \cdots & a_1
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
0 \\
0 \\
0 \\
\vdots \\
1
\end{bmatrix}
\]

We dropped the time dependency of \(X\) and \(\dot{X}\) as a notational convenience.

From (60) and (61) we can write

\[
y(t) = b_1 x_1(t) + b_2 x_2(t) + \cdots + b_m x_m(t)
\]

or

\[
y(t) = C^T X
\]

where

\[
C^T = [b_m \ b_{m-1} \ \cdots \ b_i \ 1 \ 0 \ \cdots]
\]

\(C^T\) has \(n\) elements. Equations (66) through (73) constitute a state variable representation of the transfer function, \(G(s)\).

2.3.2 z-transform method

For the z-transform method we find the z-domain equivalent of \(G_{OL}(s)\) and use something like the Matlab filter function to model the resulting z-domain transfer function, \(G_{OL}(z)\). There are several methods of deriving \(G_{OL}(z)\) from \(G_{OL}(s)\) such as the impulse invariant technique, the step invariant technique and variations on the bilinear transform. We
will use one of the bilinear transform techniques. With this technique, we derive \( G_{ol}(z) \) from \( G_{ol}(s) \) by replacing \( s \) with

\[
s = \frac{2z - 1}{Tz + 1}
\]  

(74)

where \( T \) is a sample period equivalent to the update period used in differential equation solvers such as Runge-Kutta. A bilinear transform function is included in the Matlab Signal Processing and Controls toolboxes. For those who do not have these toolboxes, a simplified bilinear transform routine is included in the appendix. The details of how to use the \( z \)-transform technique will be illustrated via example in Section 2.2.4.2.

2.3.4 An Example

To illustrate the design and modeling methods described in Section 2.2.3, we will consider a specific example. We will use the \( G_{ol}(s) \) we derived earlier. Specifically, we use (see (48))

\[
G_{ol}(s) = \frac{\alpha_n^2}{s(s + 2\zeta \omega_n)}.
\]  

(75)

We will choose a closed loop bandwidth of \( f_c = 1 \) Hz and a damping ratio of \( \zeta = 1/\sqrt{2} \). This gives \( \omega_c = 2\pi f_c = 2\pi \) rad/s and from (45)

\[
\omega_n = \omega \sqrt{(1 - 2\zeta^2) + 2\zeta^2 - 1} = \omega_c = 2\pi
\]  

(76)

With this we get

\[
G_{ol}(s) = \frac{\omega_n}{s(s + 2\zeta \omega_n)} = \frac{(2\pi)^2}{s(s + 2(1/\sqrt{2})(2\pi))} \approx \frac{39.5}{s^2 + 8.9s}.
\]  

(77)

The resulting block diagram is shown in Figure 12. We will discuss \( K_R \) shortly. We note that \( G_{ol}(s) \) has a single pole at \( s = 0 \). This means that the tracker is Type 1 and we expect it to track a step input with zero steady state error. Also, it should track a ramp input with a non-zero, but finite, steady state error and we expect that it will not be able to track a quadratic or higher order input.
2.3.4.1 State variable method

To derive a state variable form we need to rewrite \( G_{ol}(s) \) in a form consistent with (53). In particular we need to write

\[
G_{ol}(s) = K_{ol} G(s)
\]

where \( G(s) \) has the form of (53). For our example this results in

\[
G(s) = \frac{1}{s^2 + 8.9s + 0} = \frac{1s^0}{s^2 + a_1s + a_2}.
\]

Comparing (79) to (53) we note that \( m = 0, \ n = 2, \ a_1 = 8.9 \) and \( a_2 = 0 \). Since \( n = 2 \), \( A \) is a 2×2 matrix given by

\[
A = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -8.9 \end{bmatrix}.
\]

\( B \) is a 2-element column vector of the form

\[
B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
\]

and \( C^T \) is a 2-element row vector of the form

\[
C^T = [1 \ 0].
\]

With this we can write the state variable representation as.

\[
\dot{X} = AX + Bu(t), \quad y(t) = C^T X
\]

or

\[
\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -8.9 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t), \quad y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}.
\]
We can use this to redraw the block diagram in the form shown in Figure 13. Note that we replaced \( u(t) \) with \( e(t) \) since the input to \( G_{OL}(s) \) is \( e(t) \). We also added the \( K_{OL} \) block since the equations of (84) are a state variable representation of \( G(s) \), not \( G_{OL}(s) \). Recall that \( G_{OL}(s) = K_{OL}G(s) \).

![State variable Block Diagram For Example](image)

Figure 13– State variable Block Diagram For Example

A flow diagram illustrating how to simulate the tracker using the state variable method is contained in Figure 14. We start by defining \( A, B, C, K_{OL}, K_R \), and the integration step size, \( dt \).

We must take care when setting \( dt \). If it is too small, the simulation run time may be unacceptably large and if it is too large the simulation results will be wrong because we will have violated the Shannon sampling theorem. A rule of thumb is to set the initial value of \( dt \) to \( 1/10 \) the smaller of

1. The reciprocal of the closed loop bandwidth and
2. The reciprocal of the magnitude of the largest open loop pole divided by \( 2\pi \).

In our case, the closed loop bandwidth is 1 Hz and the magnitude of the largest open loop pole is \( |s| = 8.9 \). The latter would yield a value of \( 8.9/2\pi = 1.414 \). Based on this, we would start with \( dt = (1/1.414)/10 = 0.0707 \) s, which we would simplify to \( dt = 0.05 \) s. We would run the simulation with the initial \( dt \) and then with a smaller \( dt \). If the results are similar, we could use the initial \( dt \). We could also run with a larger \( dt \) and, if the results were similar, keep the larger \( dt \). In this example, we started with \( dt = 0.05 \) but found that \( dt = 0.02 \) was better.

Once \( dt \) is defined, we can compute the time array and the input, \( x(t) \). We also set the initial state, \( X(0) \), to zero and define the input, \( x(t) \). We also set the initial error, \( e(0) \) and the initial output, \( \hat{x}(0) \), to zero.
The first operation in the iterative loop is to compute the state one $dt$ in the future. The block diagram shows this being done with a routine called solver. This could be a Runge-Kutta routine or some other differential equation solver. A specific example is the Matlab ode45 routine. The inputs to the solver are the state variable equation, the integration step size, $dt$, the initial state, $X_{old}$, and value of the input at the current time, $e$. The output is the new state vector, $X_{new}$.

In the next block, $X_{old}$ is set to $X_{new}$ and will serve as the initial condition on the next pass through the loop.

The next two blocks compute the tracker output, $x_{hat}$, and the error, $e$, for the next pass through the loop. As a note, the input used to compute $e$ (from $e = x - x_{hat}$) is computed at the next time step, $t_{new}$.

The Matlab script of Figure 15 is an implementation of the methodology just discussed. The script uses the Matlab ode45 differential equation solver. The state variable equation is input to the ode45 routine using the anonymous function approach and the function, TF, is the function that contains the state variable equation. The protocol for using the ode45 function can be obtained by typing “help ode45” at the Matlab command prompt.

Plots of the tracker output will be presented after we discuss how to implement the tracker using the z-transform method.
% Script to implement analog tracker using state variable approach
clear;close all
dt = 0.02; % Sample period
t = [0:dt:3]; % define the time array
KR = 1; % Set KR = 1

% Select an input type
input = 's'; % s, r, q for step, ramp or quadratic
if input == 'q'
    x = t.^2;
    titl = 'quadratic input';
elseif input == 'r'
    x = t;
    titl = 'ramp input';
else
    x = ones(size(t));
    x(1) = 0; % Set the initial value to zero
    titl = 'step input';
end

% establish the output arrays
xhat = zeros(size(t));
delx = zeros(size(t));

% Define the initial state and the output matrix
% A and B are defined in the function TF
Xold = [0; 0];
C = [39.5 0];

% Run the loop
for kt=1:length(t)-1
    % implement the track filter. X is the state vector
    [t1 X] = ode45(@(tt,X) TF(tt,X,u),[0 dt],Xold);
    Xold = X(end,:); % save the state for the initial condition
    vx = C*Xold; % compute the output of the track filter
    xhat(kt+1) = vx/KR; % compute xhat
    delx(kt+1) = x(kt+1) - xhat(kt+1); % compute the error
    e = KR*delx(kt+1); % compute e
end

% Plot the output
fh1 = figure(1);
subplot(211)
plot(t,xhat,t,x,'r','LineWidth',2)
ylabel('Position')
xlabel('Time (s)')
legend('xhat','x')
grid on

subplot(212)
plot(t,delx,'LineWidth',2)
ylabel('Error')
xlabel('Time (s)')
grid on

function Xd = TF(t,X,u)
% Function to implement the state variable equation for the loop filter
A = [0 1; 0 -8.9];
B = [0; 1];
Xd = A*X + B*u;
return

Figure 15 – Matlab Script for State Variable Method
2.3.4.2 z-transform method

For the z-transform method we also start with Figure 12. However, in this case, we use the bilinear transform to convert $G_{ol}(s)$ to $G_{ol}(z)$. In practice, this is easily done using Matlab’s bilinear function or the bilins2z function in the appendix. We included the appendix function because the Matlab bilinear function is part of the signal processing and controls toolboxes, which not everyone has.

Once we have $G_{ol}(z)$ we use the Matlab filter function to implement $G_{ol}(z)$. We will discuss this shortly. For now, we take a tangent and show how the bilinear transform is used to find $G_{ol}(z)$ from $G_{ol}(s)$ for the specific $G_{ol}(s)$ in Figure 12.

Recall that we derive $G_{ol}(z)$ from $G_{ol}(s)$ by replacing $s$ with the bilinear transform given by (74). For our case, we have

$$G_{ol}(s) = \left. G_{ol}(s) \right|_{s = \frac{z - 1}{T}} = \frac{K_{ol}}{s + \frac{T}{z - 1}} = \frac{K_{ol}}{s + \frac{2}{z - 1}}$$

$$= \left(\frac{T}{2}\right)^2 K_{ol} \left(\frac{z + 1}{z - 1}\right)^2 = \left(\frac{T}{2}\right)^2 K_{ol} \left(\frac{z^2 + 2z + 1}{z^2 - 1}\right)$$

If we use this in the block diagram of Figure 12, that diagram would look like Figure 16.

![Figure 16 – Block Diagram of z-transform Method](image)

A logic diagram illustrating how to simulate the tracker using the z-transform method is contained in Figure 17. As with the state variable method, we start by defining the various parameters. To be consistent with the form required by the bilinear (bilins2z) function we need to cast $G_{ol}(s)$ as a ratio of polynomials. For our example this gives

$$G_{ol}(s) = \frac{39.5}{s^2 + 8.9s + 0}$$

Where we included the 0 in the denominator because we need to include all coefficients of $s^n$. Once we have the numerator and denominator coefficients of $s$ for $G_{ol}(s)$, we use the bilinear transform function to compute the numerator and denominator coefficients of $z$ for $G_{ol}(z)$.
The conditions on the choice of $T$ in the $z$-transform method are the same as for choosing $dt$ in the state variable method.

The computation loop for the $z$-transform method is similar to the state variable method except that the Matlab filter function is used to compute the output of $G_{OL}(z)$. As a note, the filter function requires an initial state vector (as did the state variable method) and returns the updated state vector. The lengths of these vectors is equal to the order of the denominator of $G_{OL}(s)$ (2 in our example). As with the state variable method, we start with an initial state vector of zero. A Matlab script illustrating the the $z$-transform method is contained in Figure 18.
% Script to implement analog tracker using bilinear/filter approach
clear;close all
T = 0.02; % Set the sample period to 0.02 sec
dens = [1 8.9 0]; % Denominator polynomial coefficients - s-domain
nums = 39.5; % Numerator polynomial coefficients - s-domain
KR = 1; % Set KR = 1;
% Find z-domain transfer function using bilinear transform
[numz,denz] = bilinear(nums,dens,1/T);
t = [0:T:3]; % Set-up a time array
% Select an input type
input = 's'; % s, r, q for step, ramp or quadratic
if input == 'q'
    x = t.^2;
    titl = 'quadratic input';
elseif input == 'r'
    x = t;
    titl = 'ramp input';
else
    x = ones(size(t));
    x(1) = 0; % Set the initial input to zero
    titl = 'step input';
end
% establish the output arrays
xhat = zeros(size(t));
delx = zeros(size(t));
% Execute the loop
zi = [0 0]'; % set the initial condition for the filter function
e = 0; % set the initial track filter input
for kt = 1:length(t)-1
    [vx,zf] = filter(numz,denz,e,zi); % implement the track filter
    zi = zf; % save the initial conditions for the next iteration
    xhat(kt+1) = vx/KR; % compute xhat
    delx(kt+1) = x(kt+1) - xhat(kt+1); % compute the error
    e = KR*delx(kt+1); % compute e
end %loop
% Plot the output
fh1 = figure(1);
subplot(211)
plot(t,xhat,t,x,'r','LineWidth',2)
ylabel('Position')
xlabel('Time (s)')
grid on
title(titl)
subplot(212)
plot(t,delx,'LineWidth',2)
ylabel('Error')
xlabel('Time (s)')
grid on

Figure 18 – Matlab Script for z-transform Method

2.3.4.3 Example plots

Figures 19, 20 and 21 contain plots of the simulation output for the cases where the input is a step, a ramp and a quadratic. The top graph contains plots of the input \( x(t) \) - in red and
the tracker output ($\hat{x}(t)$ - in blue). The bottom graph contains plots of the error ($\Delta x(t) = x(t) - \hat{x}(t)$). It will be noted that the tracker follows a step input with zero steady state error. For the ramp input there is a constant steady state error and for the quadratic input the error increases with time. These are the expected behaviors because $G_{ol}(s)$ has a single pole at the origin of the $s$ plane, which means that we have a Type 1 servo. The settling time is about one second, which is consistent with the one-sided loop bandwidth of 1 Hz, and there is a small overshoot in the step response and the error plots, which is consistent with our selection of $\zeta = 1/\sqrt{2}$.

Figure 19 – Simulation Output for Step Input
Figure 20 – Simulation Output for Ramp Input

Figure 21 – Simulation Output for Quadratic Input